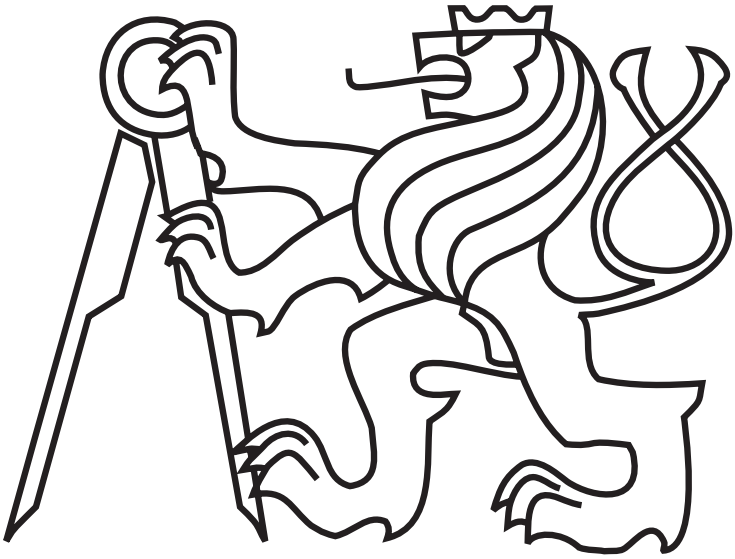
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR THESIS



Viktor Kozák

Local planning for a mobile robot

Department of Cybernetics

Thesis supervisor: **RNDr. Miroslav Kulich, Ph.D.**

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Viktor K o z á k

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Metody lokálního plánování pro mobilní robot

Pokyny pro vypracování:

1. Seznamte se s metodami pro lokálního plánování pro terestriální mobilní robot.
2. Implementujte vybranou metodu lokálního plánování.
3. Vytvořte rozhraní implementované metody do systémů Player/Stage a ROS.
4. Experimentálně ověřte funkčnost a vlastnosti implementované metody v simulátoru i s reálným robotem a získané poznatky zdokumentujte.

Seznam odborné literatury:

[1] Seder, M.; Macek, K.; Petrovic, I.: An integrated approach to real-time mobile robot control in partially known indoor environments. Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE, pp.6, 6-10 Nov. 2005, doi: 10.1109/IECON.2005.1569176
[2] Seder, M.; Petrovic, I.: Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. Robotics and Automation, IEEE International Conference on, pp. 1986-1991, 10-14 April 2007.
[3] Robot Operating System, ros.org, accessed 26.6.2013.
[4] Vaughan, R. T.: Massively multi-robot simulations in Stage. Swarm Intelligence, 2(2-4): 189-208, 2008.

Vedoucí bakalářské práce: RNDr. Miroslav Kulich, Ph.D.

Platnost zadání: do konce letního semestru 2013/2014

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 28. 6. 2013

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Viktor K o z á k

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: Local Planning for a Mobile Robot

Guidelines:

1. Get acquainted with local planning methods for a terrestrial robot.
2. Implement a chosen local planning method.
3. Create interfaces for Player/Stage and Robot Operating System (ROS).
4. Verify experimentally functionality and properties of the implemented method in a simulator and with a real robot. Describe and discuss the obtained results.

Bibliography/Sources:

[1] Seder, M.; Macek, K.; Petrovic, I.: An integrated approach to real-time mobile robot control in partially known indoor environments. Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE, pp.6, 6-10 Nov. 2005, doi: 10.1109/IECON.2005.1569176
[2] Seder, M.; Petrovic, I.: Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. Robotics and Automation, IEEE International Conference on, pp. 1986-1991, 10-14 April 2007.
[3] Robot Operating System, ros.org, accessed 26.6.2013.
[4] Vaughan, R. T.: Massively multi-robot simulations in Stage. Swarm Intelligence, 2(2-4): 189-208, 2008.

Bachelor Project Supervisor: RNDr. Miroslav Kulich, Ph.D.

Valid until: the end of the summer semester of academic year 2013/2014

L.S.

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, June 28, 2013

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the sources (literature, software, etc.) in accord to the Methodological Instructions on adherence to ethical principles in the preparation of university theses.

In Prague on..... .....

Acknowledgements

I would like to thank my thesis supervisor RNDr. Miroslav Kulich, Ph.D for his professional leadership, advice on my work and for his patience with me. I would also like to thank my family and all the people who supported me during my years at the university.

Abstrakt

Tato bakalářská práce se zabývá porovnáním různých metod lokálního plánování pro mobilní robot. V rámci práce byla pro tento účel implementována metoda Dynamic Window pro systém Player/Stage. Metoda byla ověřena simulacemi na několika mapách, a otestována pro řízení reálného robota na systému SyRoTek, umístěném v laboratoři na Katedře kybernetiky. Získané výsledky byly porovnány s výsledky stávajících metod. Byly nalezeny nejlepší konfigurace pro použití daných metod na systému SyRoTek.

Abstract

The subject of this bachelor thesis is a comparison of local planning algorithms for mobile robot. As a part of the thesis Dynamic Window method was implemented in the Player/Stage system. The Method was tested on various simulations and for control of a real robot in the SyRoTek system, located in a laboratory at the Department of Cybernetics. Results gained from the experiments were compared with results of state-of-the-art methods. Best configurations for the use of given methods with the SyRoTek system were found.

Contents

Chapter 1

Introduction

Local planning algorithms are an essential part of today's mobile robot and autonomic vehicle control. While the global planning decides the route of the robot based on the initial data given to the planner, the local planning is a real-time motion control, based on the feedback from sensors. It's purpose is to keep the robot on an optimal track, following the global planner and to avoid unexpected obstacles, making it a fundamental part for safe robot navigation.

The thesis aims to compare three local planning methods. The Dynamic Window Approach(DWA), Enhanced Vector Field Histogram(VFH+) and Smooth Nearness-Diagram(SND) algorithms. It is focused on implementation of Dynamic Window Approach based on [1].

The goal of the thesis is to build a functional local planner based on DWA and find out whether is it suitable for control of a mobile robot equipped with a laser range finder. For that purpose the algorithm was tested on various maps in the Player/Stage system[2] and on several maps with a real robot in the SyRoTek system[3]. After completion of the DWA based algorithm a huge set of simulations was completed and the results were compared with the results of VFH+ and SND algorithms in the same environment.

As a part of the thesis a large variety of parameter configurations was tested, for the use with the navigation algorithms. The thesis aims to determine the best configurations for the use with the SyRoTek system.

Chapter 2 gives the reader a short introduction to the topic. It explains the meaning of local planning algorithms and covers basic descriptions of each approach and the main differences between them.

Chapter 3 describes the frameworks used for simulations and experiments. It presents the SyRoTek system and its characteristics. It also introduces the Player/Stage framework and explains its uses and possibilities for robot control.

Chapter 4 describes the process of creating the algorithm and implementing it in the Player/Stage framework for further use together with all the obstacles faced during the implementation.

Chapter 5 presents results of all simulations in Player/Stage program and experiments on the real robot in the SyRoTek arena. In the first part are results for the DWA algorithm created in this thesis and the SND and VFH+ algorithms from Player/Stage library. The last part of the chapter is dedicated to the comparison of results for the created algorithm and algorithms already implemented. It discusses the advantages, disadvantages and possible uses for each algorithm.

Chapter 2

Motion planning

Motion planning is a term often used in robotics. It can be described as a task to produce a continuous motion that connects the initial state of a system with the required state of the system.

In [4], a simpler version of motion planning is described as the *Piano Mover's Problem*. Where the model of a house and a piano is given as input to the algorithm and the algorithm must determine, how to move the piano from one room to another without a collision. Now, if we simply exchange the piano with a mobile robot, we can get the basic image of the problem described in this thesis.

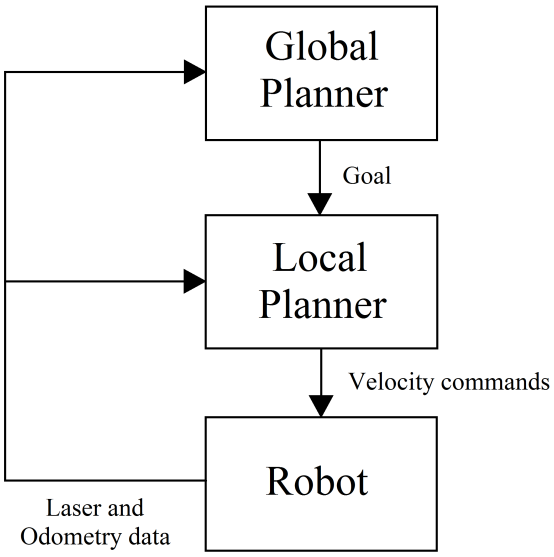


Figure 2.1: Scheme of a robot control system.

2.1 Global and local planning

Planning the trajectory for a mobile robot can be separated into two tasks. The global planning, which usually works with a full or a partial map of the environment and decides, which route to take to get to the goal and the local planning, which works with a real-time feedback from the environment surrounding the robot and its purpose is to keep the robot safely on the path chosen by the global planner and avoid any potential obstacles on the way.

An illustration of a communication structure of the navigation mechanism is shown in Fig. 2.1. During the process, the robot sends data from the laser and odometry sensors to the global and local planners. The global planner compares the odometry information with the desired goal position and sends information to the local planer. Local planner computes desired velocities suitable for the optimal trajectory and sends a velocity command to the robot.

2.1.1 Global planning

The global planning algorithm usually gets a map of the environment, the parameters of the robot, the starting point **A** and the point of the required destination **B**. Its purpose is to generate additional points (P_1,P_2,...), creating the trajectory between **A** and **B** on which it's possible for the robot to move in a direct trajectory from point P_i to point P_{i+1} without colliding into any of the known obstacles.

Although global planning algorithms are essential for automatic motion planning, for the purpose of this thesis the global planning algorithm was substituted by a human intervention and the trajectory points were set manually.

2.1.2 Local planning and obstacle avoidance

Unlike the global planning which uses the initial data given to the planner, local planning and obstacle avoidance work with real-time data gathered by sensors directly from the surroundings of the robot. It is commonly used in unmanned vehicles and autonomous robots.
Obstacle avoidance is the task of maintaining a non-collision position of the robot. The task of local planning is to determine the best trajectory based on information from the global planner, and information from sensors.

2.2 Local planning algorithms used in the thesis

For the purposes of this thesis three methods of local planning were used, each one of them based on a different concept. Methods were chosen for the ability to control a mobile robot equipped with a laser range finder.

2.2.1 Vector Field Histogram Plus

The Vector Field Histogram Plus(VFH+) is a method based on [5]. It is an improved version of the original Vector Field Histogram(VFH) method, developed for real-time motion planning in 1991, introduced in [6]. VFH is one of the most popular local planners currently used in mobile robotics.

This method uses a histogram grid for representation of the robot's environment. In the first step a two-dimensional map grid is created using data from the robot's range sensors. Reducing the map grid around the robot's current location, one-dimensional *polar histogram* is created. Then in the last stage, the algorithm selects the most suitable direction based on the polar histogram and a cost function.

An example of a polar histogram can be seen in following figures. In Fig. 2.2 we can see *polar obstacle density*(POD) represented in smoothed polar histogram **H'(k)**, where $0°$ is the direction the robot is facing. In the picture the peeks **A**, **B** and **C** are the result of three obstacles located near the robot. Smoothed polar histogram typically has "peeks", sectors with high POD and "valleys", sectors with low POD. Any valley containing sectors with POD below a certain threshold can be called a candidate valley. If there are more candidate valleys, the VFH algorithm selects the valley that most closely matches the direction to the target. Then it selects the most suitable sector within that valley and commands the robot. In Fig. 2.3 is the same polar histogram as in the previous picture, shown in the map of the environment surrounding the robot. In this map the robot is facing the target and we can see obstacles **A**, **B** and **C**, represented as black squares, and the POD represented by the grey areas surrounding the robot. Both pictures 2.2 and 2.3 are based on original pictures in [5] but were redrawn for better quality.

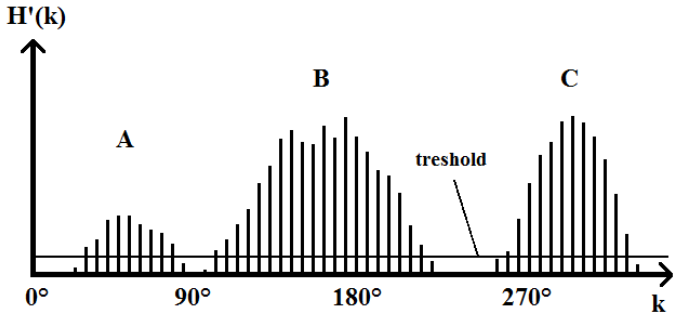


Figure 2.2: Example of a smoothed polar histogram.

In the year 1998, several improvements were made to the VFH method. The updated method was presented in [6] and named as VFH+ (sometimes referred to as "Enhanced VFH) and that is the version of VFH used for simulations in this thesis.

In the VFH+ method, several improvements were made over the original VFH method. The most important ones are listed here.

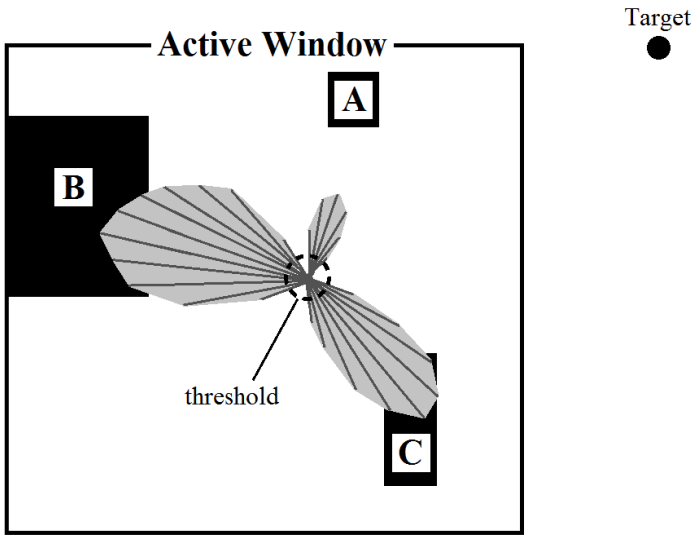


Figure 2.3: Illustration of the polar histogram from Fig. 2.2 shown in polar form overlaying part of the environment.

- **Threshold hysteresis**
  The original VFH method displayed an indecisive behaviour, when used in an environment with several narrow openings. As the opening could alternate several times between an open and blocked state during a few sampling times. During such situation, the robot's heading would alternate several times between this and another opening. By using threshold hysteresis, this problem was solved and the robot trajectory became smoother and more reliable.

- **Size of the robot**
  The VFH+ takes the width of the robot into account, by enlarging the obstacle cell, making it easier to implemented the method for robots of different sizes. This can be seen in Fig. 2.4, where the cell is enlarged by a required obstacle avoidance distance from the center of the robot $r_{r+s} = r_r + d_s$, where r_r is the robot radius and d_s is the minimum distance between the robot and the obstacle.

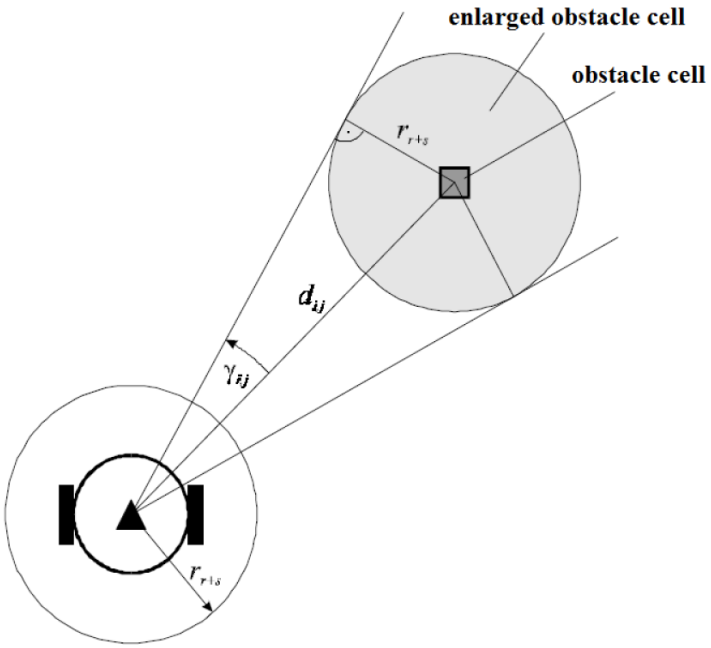


Figure 2.4: Figure describing the use of enlarged obstacle cell. The picture was taken from [6].

- **Dynamics and kinematics of the robot**
  In the original VFH method the dynamics and kinematics of the robot were neglected. The method was assuming that the robot is able to change its direction of travel instantly, as shown in Fig. 2.5a. Such assumption is impossible due to kinematics of the robot.
  The VFH+ method uses a simple approximation suitable for most mobile robots. It assumes that the robot's trajectory is based on circular arcs, as shown in Fig. 2.5b. This approximation was proved to improve the algorithm.

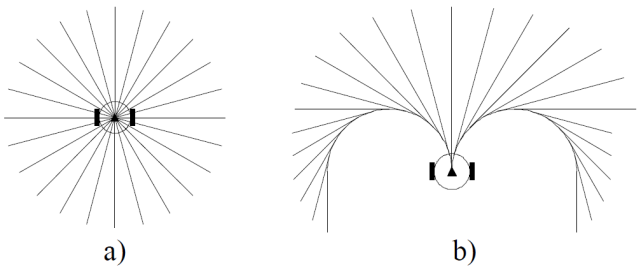


Figure 2.5: Approximation of trajectories: a) without dynamics, b) with dynamics. The picture was taken form [6].

2.2.2 Smooth Nearness-Diagram

The Smooth Nearness-Diagram(SND) method used for simulations is an improvement of the original Nearness-Diagram(ND) presented in [7]. The SND algorithm removes oscillatory patterns which occurred in ND algorithm and improves the overall driver performance. The SND method was introduced in 2008 in [8].

The SND algorithm introduces the concept of *gaps*. Gaps are discontinuities in the depth of obstacles around the robot which indicate potential paths into occluded areas of the environment. By navigating based on gaps, SND can avoid local trap situations without the computational load of determining which areas of the environment are connected.

The idea of gaps can be explained using Fig. 2.6. A gap occurs at an angle where two contiguous depth measurements are separated by more than the robot diameter r, or one of the measurements returns no obstacle in range. The first type of gap occurs at (a) in Fig. 2.6, the second at (b). We can distinguish between a left and right gap. Left gap means that the closer measured obstacle is on the left side of the gap, as in (a) in Fig. 2.6, it indicate that there may be an occluded area on the left. The opposite holds for right gaps.
By the pairs of consecutive gaps we can define different regions in the environment. A navigable region(a "valley") is distinguished by either a left gap on its left side, a right gap on its right side, or both. After assembling all the valleys surrounding the robot, all the gaps are compared against the heading provided by the global planner. The valley containing the gap with the best heading, V_{best}, is determined and selected for further use.

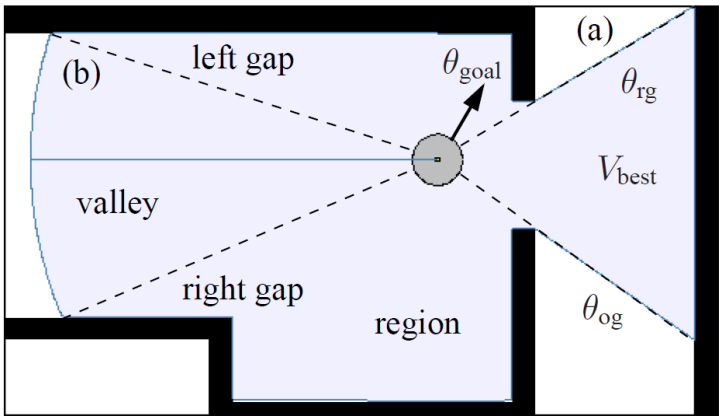


Figure 2.6: A graphic representation of valleys and gaps used in the SND method. The picture was taken from [8].

The original ND method then determines the desired trajectory based on the two closest obstacles and the width of V_{best}. Here is the main difference between the original ND method and the new SND method.
The SND method measures a threat possessed by each of the obstacles. An obstacle is considered a threat if it lies within the safety distance of the robot. Where the treat measure increases

as the obstacle gets closer to the robot. Using the threat measurements of each obstacle, a deflection from the desired heading is computed and given to the algorithm. This computation proved to remove some oscillatory patterns and improve the driver's performance in narrow paths.

2.2.3 Dynamic Window Approach

The Dynamic Window Approach(DWA) is a local obstacle avoidance method introduced in 1997 in [9]. This method is directly connected with the dynamics of the robot. It takes into account limitations of the velocities and accelerations of the robot.

The algorithm is divided into two main components. In the first part, search space is generated and in the second part, an optimal path is chosen from the generated search space.

Generating a valid search space

Valid search space is computed directly from the limitations of the velocities and accelerations of the robot. A two dimensional search space is based around the current linear and angular velocity(v_c and w_c). Using the acceleration parameters of the robot, the maximal and minimal velocities are computed as:

$$v_{max} = v_c + v_a, v_{min} = v_c - v_b \tag{2.1}$$

and

$$w_{max} = w_c + w_a, w_{min} = w_c - w_b \tag{2.2}$$

Where v_a and w_a are maximal translational and rotational accelerations and v_b and w_b are maximal decelerations executable by motors. According to the computation power of the robot and the requested precision of the algorithm, a number of possible translational(N_v) and rotational(N_w) velocities is chosen from the interval between the maximal and minimal velocities. Of course, the interval is restricted by the maximum velocity of the robot and its maximum turning rate. Combining these velocities a number of N_vxN_w velocity pairs is created.

Each velocity pair(v, w) is represented by a circular arc with the starting point in the center of the robot. It's radius is calculated as $r = \frac{v}{w}$ and the length of the arc is set as v. This representation is called the dynamic window. All curvatures outside this dynamic window cannot be reached by the robot in the next step, and thus are not considered for obstacle avoidance. Each trajectory is then compared with readings from the laser range finder. The trajectory is considered safe if the robot is able to stop before colliding with any object along the path.

In Fig. 2.7 is a graphic representation of the search space. For this example 5 linear and 7 angular velocities are set and the field of 35 pairs is created. Each pair is represented by a circular arc with given length and radius. We can notice three different colors used for the representation of the curvatures. The red color represents the expected trajectory for current

velocity parameters of the robot. The blue color represents non-safe trajectories which would cause a collision with some of the obstacles. Finally, the green color represents collision-free trajectories. A combination of all collision-free trajectories creates the valid search space which is later used in the algorithm, for the choice of the optimal path.

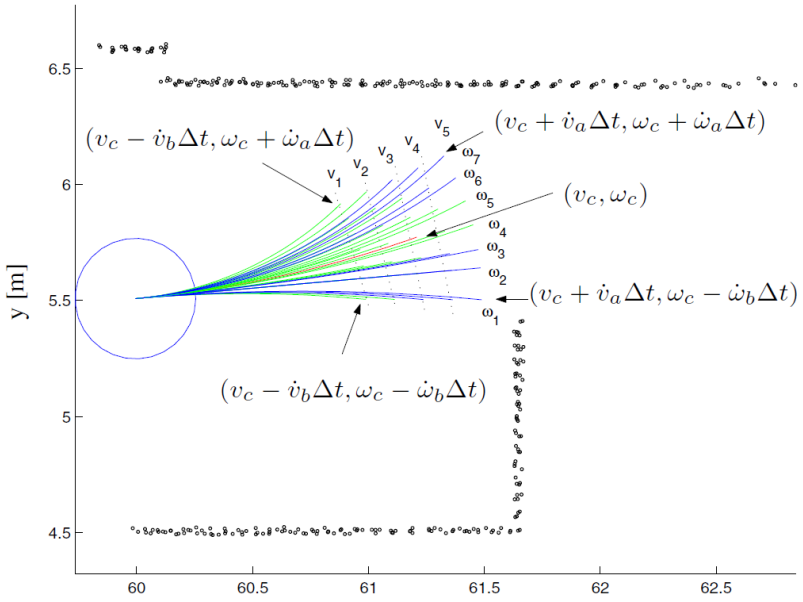


Figure 2.7: Graphic representation of possible robot trajectories. The picture was taken from [1].

Selection of the optimal trajectory

The selection of the optimal trajectory is based on three basic attributes, which differ for each velocity pair. The angular attribute ϑ_{ang}, the velocity attribute ϑ_{vel} and the clearance attribute ϑ_{clear}.

The angular attribute is telling us, how much is the direction of a particular trajectory similar to the global goal position. The velocity attribute is chosen such, that higher linear velocities are preferred. The clearance attribute is chosen in favour for trajectories further away from potential obstacles.

To improve the robot performance, an adjustable parameter λ is given to each attribute. Using these parameters, the behaviour of the algorithm can later be easily changed based on the requirements for the robot's behaviour or it can be set differently for different environments. For each velocity pair a weighted sum is computed as follows:

$$\Gamma(v, w) = \lambda_{ang}\vartheta_{ang} + \lambda_{vel}\vartheta_{vel} + \lambda_{clear}\vartheta_{clear} \tag{2.3}$$

With the highest sum, the optimal trajectory is chosen and the best velocities v_{best} and w_{best} are set as the output of the algorithm.

Chapter 3

Robotic frameworks

In this work several robotic frameworks were used. In the first part the Player/Stage program was used to create and test the DWA algorithm. After that the algorithm was tested in the SyRoTek system on the real robot, together with other algorithms already implemented in Player/Stage. Basic information about these programs are presented in this chapter, but for more information about the projects, visiting their websites is recommended.

3.1 Player/Stage

The Player project creates free software tools and enables research in robot and sensor systems. Nowadays Player is probably one of the most widely used open-source robot control interfaces in research and post-secondary education. Its components include the Player robot device interface, Stage robot simulator and Gazebo 3D robot simulator. Released under the GNU General Public License, all code from the Player/Stage project is free to use, distribute and modify. In this thesis Player is used for the control of the robot and Stage is used for simulations. The description is based on information attained from the official website of the Player project [10].

3.1.1 Player

Player provides a network interface to a wide variety of robot and sensor hardware and it can be easily used for control of a real physical robot or simulated robot in Stage. Player's client/server model allows robot control programs to be written in any programming language. It also allows the control programs to run on any computer with a network connection to the robot.

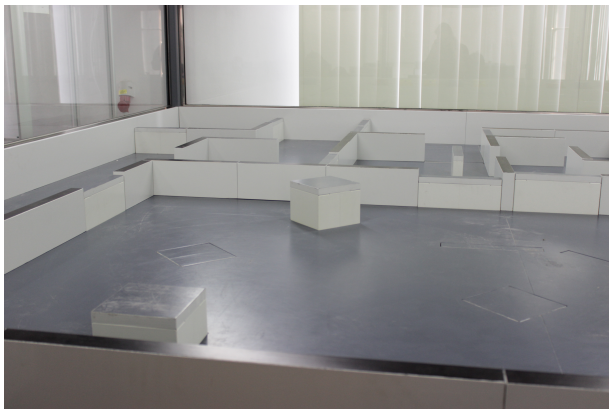
3.1.2 Stage

Stage can simulate from one up to hundreds of robots at a time in two-dimensional environment. It also provides various sensor models, including sonar, laser range finder, odometry and others. Typically, using little or no modification, Player clients, developed using the Stage simulator, will work with the real robots.

3.2 SyRoTek system

The SyRoTek system("System for robotic e-learning") is used at Czech Technical University in Prague. The system allows users to remotely control a multi-robot platform in a dynamic environment allowing them to develop their own algorithms and monitor their behaviour online during real experiments. The description is based on information attained from the official website of the SyRoTek system [11].

The SyRoTek Arena is an enclosed space dedicated for robots. The Arena can be seen in Fig. 3.1. The size of the Arena is 3.5 m x 3.8 m and it consist of the robot working space and the necessary supporting subsystems (charging, lighting, visualization, etc). The robot working space is a flat area with a number of obstacles placed inside. Some obstacles can be remotely retracted, while the rest of them is fixed, however all obstacles can be manually removed in order to create various configurations. The Arena is equipped with a global localization system which estimates the robot's identity together with its position and orientation using an image processing method.



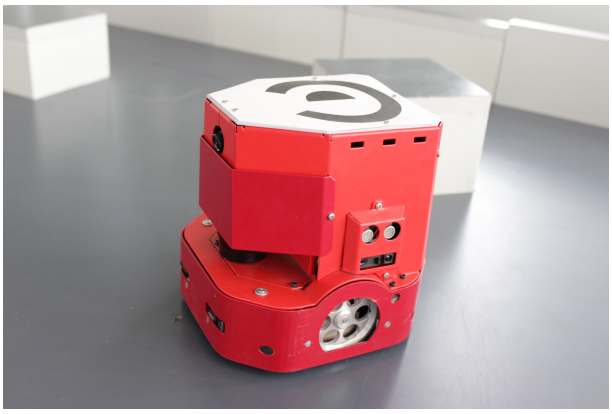


Figure 3.1: On the left picture is the SyRoTek arena and on the right picture is the S1R robot.

The system provides the S1R robots. The S1R robots can be equipped with a large variety of sensors. For the experiments in this thesis a robot equipped with a laser range-finder is used. The robot uses two wheels and a support slider for its movement. Its dimensions are (length x width x height): 174 x 163 x 180 mm. The robot can be seen in Fig. 3.1.

Chapter 4

Implementation of the Dynamic Window Approach

The implementation of Dynamic Window Approach(DWA) is a major part of this thesis. This chapter is separated into several parts, each describing a specific part of the process. The first part describes the basic algorithm for the robot navigation. It covers its creation and it gives a description of most important parts of the algorithm. The second part describes the creation of Player/Stage driver and configuration file usable for the navigation algorithm. The last part contains information about modifications made after considering simulation results and further use.

Several tools were used for the creation of the navigation algorithm, the Player/Stage framework, for which the algorithm was created, the MATLAB software used for a graphic representation of robot trajectories during the development of the algorithm and a SyRoTek environment for the use with the Player/Stage framework. The environment allowed the algorithm to be tested directly with the Stage simulator. This allowed the development of the navigation algorithm without the need of a special Player driver. The Player driver for the algorithm was written later.

The DWA driver is written and tested using the Player v.3.0.2 and Stage 3.2.2 frameworks on Ubuntu 12.10 software, so it may have to be modified if intended for use under different software environment.

4.1 Basic navigation algorithm

The function of the navigation algorithm is to read data from sensors and from the global planner as an input and to give the required future translational and angular velocities as an output. The algorithm is intended for further use with Player/Stage framework, therefore there is no restriction regarding the used programming language. Considering it's simple and universal use, the C++ language was chosen for the task.

4.1.1 Setting initial variables

The algorithm can be divided into three parts. In the first part initial variables are set. This part of the algorithm includes data readings from robot sensors, global position and global planner. It also includes the creation of basic variables used later in the algorithm.

Firstly the program establishes a connection with Player proxies and takes the initial data from them. In this part specific data are taken from the driver and set as local variables for further use. These data include such information as robot radius, maximum speed, laser specifications, and other information constant for the rest of the program. Among these data are parameters given to the driver in a configuration file by the user. The structure and function of the configuration file is closely described in Section 4.2.

After reading these time-invariant variables, the program goes into an infinite loop, until it is terminated or until it reaches the goal. In the loop, current data from sensors are read including such data as the current robot position, laser readings and current velocities, which have to be updated in every run of the algorithm. In this part the algorithm checks with the global position and the goal position. Eventually is terminated if the robot is within an acceptable distance from the goal position. From the initial data, other basic variables are computed for further use in the algorithm.

4.1.2 Creation of a valid search space

For this step maximal and minimal velocities are computed as in the equations 2.1 and 2.2 and limited by the maximum and minimum speed of the robot. Then a number of velocities is chosen from within the intervals $\{v_{min}, v_{max}\}$ and $\{w_{min}, w_{max}\}$. For the algorithm number of 12 angular and 5 translational velocities is chosen, but it can be easily changed according to the required precision or the required speed of the algorithm.

Combining these velocities an array of velocity pairs is created. For every velocity pair in the array additional parameters are computed. Namely the turning radius r, breaking distance d_b, and coordinates of the center of the circle representing the velocity pair trajectory. The radius of the circle is set as $r = \frac{v}{w}$ and the breaking distance is computed from the translational velocity and the maximal translational deceleration. The coordinates are on a line perpendicular to the direction of the robot, located in the distance equal to r. For a special case where $w = 0$, is the radius r substituted by an incomparably large number, to simulate a straight line.

To determine which velocity pairs provide a safe trajectory, on which the robot is able to stop before colliding with any object along the path, an algorithm for intersection of two circles is used. One circle is a representation of the robot's trajectory, defined by the additional parameters of each velocity pair. The second circle represents an obstacle detected by the sensor.

Circles representing the obstacles are created in a similar way as it was described in 2.2.1. Each circle has the origin in a distance d_i from the center of the robot in the corresponding angle α_i. In Fig. 2.4 the radius of each circle is set as $r_{r+s} = r_r + d_s$, where r_r is the robot

radius and d_s is the minimum distance between the robot and the obstacle. In this algorithm additional parameter r^+ is used.

Originally all the obstacles would be represented by circles with identical radius, but that is not exact. If we take a part of the environment covered by one part of the sensor, which gives us one distance reading, and illustrate it as a circular sector, it can be seen that with greater distance from the center of the robot, the sensor should cover a larger space. Therefore the radius of a circle representing the obstacle should be larger, with distance from the center of the robot. In the algorithm this problem is compensated by adding the r^+ parameter to the circle radius representing the obstacle. The r^+ parameter is computed as:

$$r_i^+ = d_i \frac{L_r \pi}{360 N_a} \tag{4.1}$$

Where d_i represents the distance of current obstacle, N_a represents the quantity of angle readings taken from the laser range finder and L_r is the range of the laser in degrees. The final circle radius for the obstacle is then $R = r_r + d_s + r^+$. An example of the obstacle representation, used in the algorithm, is shown in Fig. 4.1. To illustrate the size changes in the circles, very low measurement density is used in the picture.

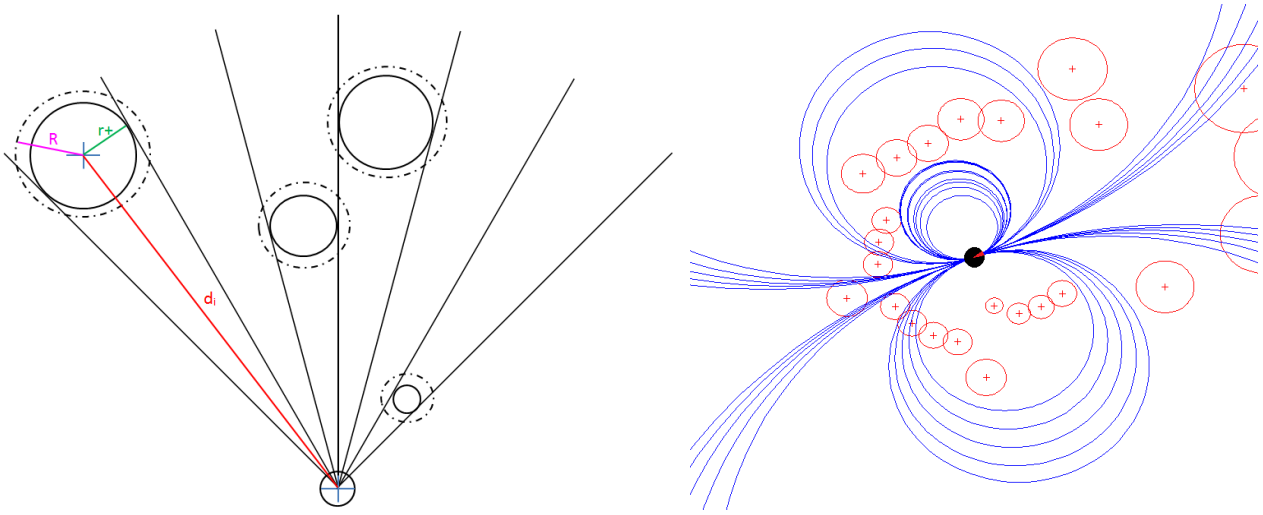


Figure 4.1: Illustration of the radius extension for different obstacles is presented on the left picture. MATLAB visualisation of robot trajectories is on the right picture.

After computing all the necessary data, the algorithm for intersection of two circles is used. Every circle representing the trajectory for a velocity pair is compared with all circles representing the obstacles. If an intersection is found, the distance of the intersection and the center of the robot is compared to the breaking distance belonging to the corresponding velocity pair. The velocity pairs with no intersection and pairs with breaking distance lower than the distance to the intersection are selected. A valid search space is created from these pairs and used further in the algorithm.

To test this part of the algorithm the MATLAB software was used for graphic visualisation of the circles representing the trajectories and obstacles. The result of the algorithm can be seen in Fig. 4.1. The robot is in the center of the figure. It is moving forward and the front part of the robot is marked with the red color. The red circles represent obstacles and the blue lines represent possible robot trajectories which are not in collision with any of the obstacles.

4.1.3 Selecting an optimal trajectory

After the creation of the valid search space, additional attributes are assigned to each velocity pair and the optimal trajectory is chosen. For the choice of the optimal trajectory three attributes are computed, each attribute within the interval $\{0, 1\}$, with 0 being the best possible option. The attribute for obstacle avoidance ϑ_{clear}, defines how far is the robots trajectory from colliding into an obstacle. The angular attribute ϑ_{ang}, defines how well the trajectory follows the desired direction to the goal.The speed attribute ϑ_{vel} ensures, that higher translational velocities are prioritized.

Each of these attributes is then multiplied by an adjustable parameter λ, that can be set by the user to modify the characteristics of robot's behaviour. A closer description of these parameters is given in 2.2.3. A weighted sum is computed for each velocity pair using the equation 2.3. Then a velocity pair with the lowest weighted sum is selected as the best choice for the optimal trajectory. The selected velocity pair is given as an output of the navigation algorithm and used in the velocity command for the motors.

4.1.4 Necessary changes of the algorithm

During the development of the navigation algorithm, few changes had to be made to improve its behaviour. Some were only minor changes in the structure of the algorithm or in parameters, but other changes highly improved the overall driver performance. After first simulations a major problem in robots behaviour occurred, the navigation algorithm constructed to prefer higher translational velocities caused the robot to avoid narrow openings and to move in undesired directions.

The problem is closely described in Fig. 4.2. The robot is moving in a straight trajectory from position **A** to **B** and then it's supposed to go to **C**. The main focus of the picture should be on the circles c_1 and c_2 representing possible robot trajectories from the **B** position.

A huge difference between the radii of the circles can be seen on the picture, while the c_2 circle represents a trajectory with a relatively small turning radius, the radius of the c_1 circle is too big and the robot would be unable to perform any sharp turns on this trajectory. That is caused by the imbalance between the maximal translational and maximal angular velocities.

The radius of the circle can be computed as $r = \frac{v}{w}$. The value of the radii is set as:

$$r_1 = \frac{v_1}{w_{max}} = \frac{1.6}{1} \tag{4.2}$$

and

$$r_2 = \frac{v_2}{w_{max}} = \frac{0.5}{1} \tag{4.3}$$

Although we can see that both circles represent trajectories with the same angular velocity, in this case the maximal possible angular velocity(w_{max}), there is a great difference in radii of the circles because each of them is connected with a different translational velocity(v_1 and v_2). This implements that if the robot cannot achieve higher angular velocity, it's turning ratio can be increased by decreasing it's translational velocity.

This issue was solved by reducing the translational velocity proportionally to the difference between the current robot heading direction and the direction to the goal. If the difference exceeds certain value, the maximal allowed translational speed of the robot is reduced.

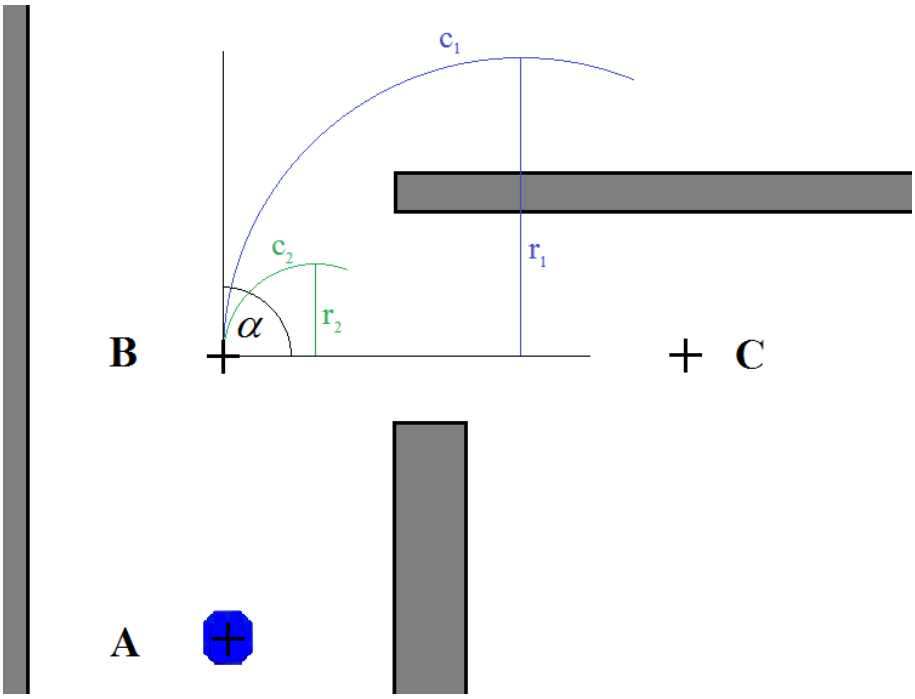


Figure 4.2: Graphic visualisation of different robot trajectories

During the experiments on the real robot a new function has been developed. The function makes the robot slow down before reaching its goal. The function can be turned off through the configuration file, while turned off, the speed of the robot may be increased on direct trajectories. The function was developed after the simulations were made, so in the simulations the robot did not slow down before the goal.

The DWA driver requires information on current velocities of the robot, but other interfaces are sometimes unable to provide such information. An on/off option was made to substitute the velocity readings by the last velocity command. During the experiments with the SyRoTek system, this option didn't seem to have negative impact on the drivers behaviour.

4.2 Driver for Player/Stage

The navigation algorithm was created and tested using the SyRoTek environment in Player/Stage. Although the environment secured a connection with Stage simulation program, it was desirable to create a new Player driver specifically for this algorithm. The new Player driver allows other users and other drivers to use the DWA algorithm. It is intended for the support of similar functions and usability as other drivers for the Player/Stage framework. It has to be able to secure a connection between the player server and users client code.

A Player driver is a tool used for communications between the physical robot and users client code. It implements standard methods for communication with proxies, listed in the Player documentation. A proxy is a Player-defined standard communication used to access various interfaces, to provide data from robot to client and vice versa.

Although most manuals found on the internet provide basic information on the matter, they proved to be insufficient for the task of creating a complex driver. The creation of the driver is mildly inspired by [12], but most of the code is written using information from the official Player/Stage web site [10], the code from the example driver provided in the standard Player distribution and the SND driver written by Joey Durham and Luca Invernizzi.

Information on how to write a driver are not included in this thesis. Such information could be find on the Player/Stage web site or in some of the manuals on the internet. This chapter presents only information connected with the developed DWA driver, problems faced during the development and information on the use of this particular driver.

When writing a driver for Player, it is possible to choose from static and plugin drivers. For the development of the driver for the DWA algorithm the plugin driver was chosen. That means that the driver is not a part of the main Player distribution code and is managed as a shared object loaded at runtime. It also means that the driver has to be compiled before use and the path to the driver has to be added in the player configuration file. Reasons for developing DWA as a plugin driver are the advantages over the static drivers. The plugin drivers are easier to build, their code can be maintained in a separate source repository and they allow easier development with faster code/compile/test cycle.

Before writing a driver, it has to be decided which interfaces will be supported by it. An interface is a pre-defined set of messages and data types for communication with a certain device or an algorithm and the Player project already has a large variety of these implemented. Interfaces for this driver are the *position2d* interface, used for position information, velocity and odometry information and position commands and the *laser* interface, used for communication with the laser.

After the selection of interfaces, a configuration file with given parameters can be created for the driver. A configuration file is a text file with the *.cfg extension. Its purpose is to define the supported drivers for the system and give details for each driver. For each supported driver required and provided interfaces are listed, and additional configuration parameters are given.

An example of a configuration file for the driver can be seen in Listing 4.1. A driver is identified by its name and for the plugin driver by a path to a shared library. The provided and required interfaces are listed. This driver provides the *position2d* interface through which it commands the robot and it requires the *position2d* and *laser* interface for information from sensors. The name of the driver and interfaces are the main parameters in configuration file. Additional parameters to adjust the characteristics of the driver can be given in the configuration file by the user. In this example robot radius and maximum speed parameters are set.

The last parameter in the example is an optional option for all the drivers, the *alwayson* option. If the *alwayson* is set as 1, then the driver will be setup when the Player server starts, without waiting for any client connection. It is useful for drivers with startup delays, and drivers used without a client. It is very useful for the development of new drivers.

```
driver
(
  name "dwdriver"
  plugin "dwdriver.so"
  provides ["position2d:1"]
  requires ["input:::position2d:0" "output:::position2d:0" "laser:0"]

robot_radius 10
max_speed 0.3
alwayson 0

)
```

Listing 4.1: An example of a configuration file.

All the parameters for the DWA driver are listed in the beginning of the "dwdriver.cc" file on the CD, together with their description. The parameters are also listed in the manual on the website created for the DWA driver [13].

With the configuration file ready, the driver can be finally written and tested. A large part of the code is a modified version of the code taken from the SND driver written by Joey Durham and Luca Invernizzi. Some changes were made only in the names of variables, functions and classes, but few bigger changes had to be made. The driver had to be adjusted for a different navigation algorithm. Functions for the reading of current velocities were added and lines for communication between the configuration file and the navigation algorithm were made, making it possible for the user to adjust robots behaviour without the need of a direct change in the inner algorithm.

Chapter 5

Experimental results

This chapter covers results from experiments with the developed algorithm. To achieve usable results, the algorithm was tested together with the SND and VFH algorithms from Player/Stage libraries.

5.1 Parameters settings

After it's development, the DWA algorithm was tested on various maps and parameter configurations with the highest success rate were chosen for further experiments. The configurations for the VFH algorithm were chosen by the same method. At first the algorithms were tested manually. After achieving satisfactory results, a broad range of parameter configurations was made, based around the parameter values found by hand. These configurations are shown in tables 5.1 and 5.2.

The configuration for the SND algorithm is the same configuration which is currently used with the SyRoTek system. The SND algorithm proved to be reliable on the SyRoTek system by previous use, and thus will be used as a milestone for the expected results.

Driver	Number of combinations	Parameter	min.	max.	step
DWA	320	robot_radius	0.10	0.13	0.01
		ValAngle	5	20	5
		ValVelocity	5	20	5
		ValObstacle	0	20	5

Table 5.1: Tested parameters for the DWA driver.

The configurations for the DWA algorithm in table 5.1 were tested on the Arena 0 map (Fig. 5.1). 10 test runs were made for each configuration and best 60 configurations, which achieved 100% success rate, were taken for the final round of experiments described in Section 5.2.

These configurations are missing the *slow* parameter, which turns on or off the function to slow down before the target. The reason for this is that the "slow down" function was developed after the simulations. That means that the robot doesn't slow down before obstacles during the simulations.

The configurations for the VFH algorithm in table 5.2 were tested on the Arena 0 map (Fig. 5.1). The 9 most successful configurations were taken for the final round of experiments described in Section 5.2.

Driver	Number of combinations	Parameter	min.	max.	step
VFH	108	cell_size	0.01	0.02	0.01
		window_diameter	10	20	10
		safety_dist_0ms	0.05	0.09	0.02
		weight_desired_dir	5	9	2
		free_space_cutoff_0ms	2000000	4000000	1000000

Table 5.2: Tested parameters for the VFH driver.

The search for ideal parameter configurations for the DWA algorithm is made on a larger scale than for the VFH algorithm. That is because the DWA algorithm is developed as a part of this thesis and it is desirable to promptly test it's full capability.

5.2 Simulations

Various simulations were made, using the Stage simulator. The simulator made it possible to create a larger variety of maps for the simulations. Thanks to the Stage simulator, any idea for a map of the environment could be easily designed in a form of a simple black and white picture or in the form of a *.world file where each obstacle is represented by its size and position.

To achieve more exact results from the simulations, 50 test runs were made for each configuration of every algorithm on every map. The simulations were made for one configuration of the SND algorithm (tbl. 5.4), 9 configurations for the VFH algorithm and 60 configurations for the DWA algorithm. The choice of the configurations is based on Section 5.1.

5.2.1 Design

While designing areas for the experiments there were three main objectives to take into consideration. First objective was to test the algorithm in narrow corridors, second to test it in wide, open space and the last one was the combination of the prior two, to test the overall behaviour. According to these objectives, several maps were designed to test the S1R robot on simulations in environments similar to the Arena in the SyRoTek system.

A short description and a schema of every map is presented. The width of the narrowest passageway in the map is included in the description, it is marked with the shortcut "n.p.". The schema contains the map of the environment with the starting position represented by a blue icon of the robot, and the goal position represented by a red icon. Some of the maps contain a secondary starting point represented by a yellow icon or a black line representing the desired trajectory of the robot.

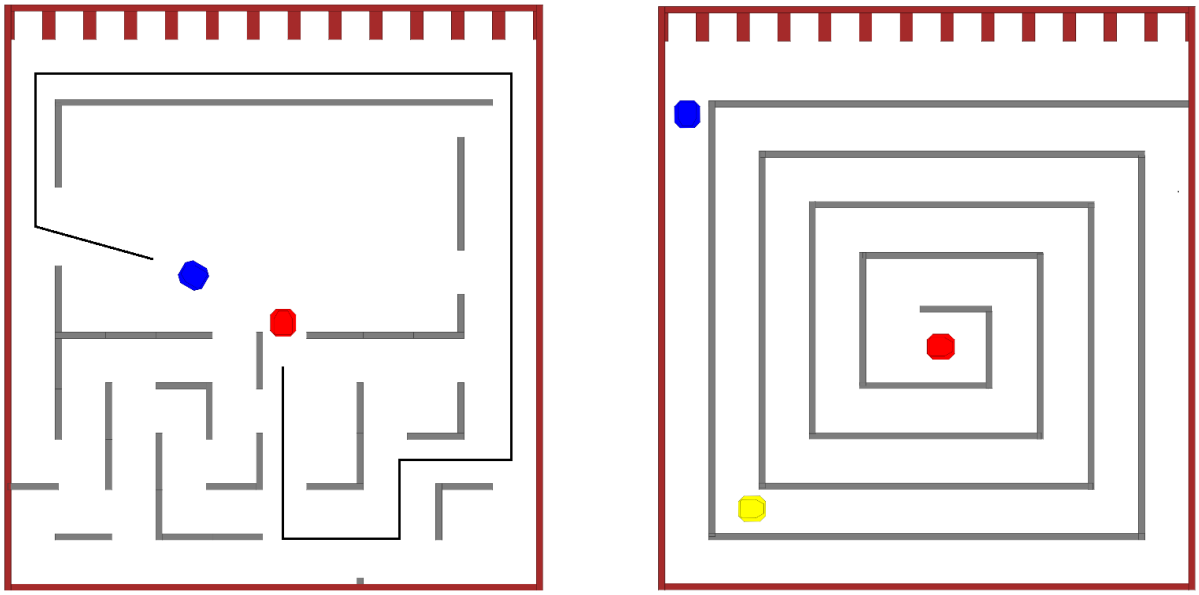


Figure 5.1: Arena 0 (n.p. = 28 cm) on the left and Arena 1 (n.p. = 28 cm) on the right. The maps are used for the simulations of the SyRoTek arena for the S1R robot.

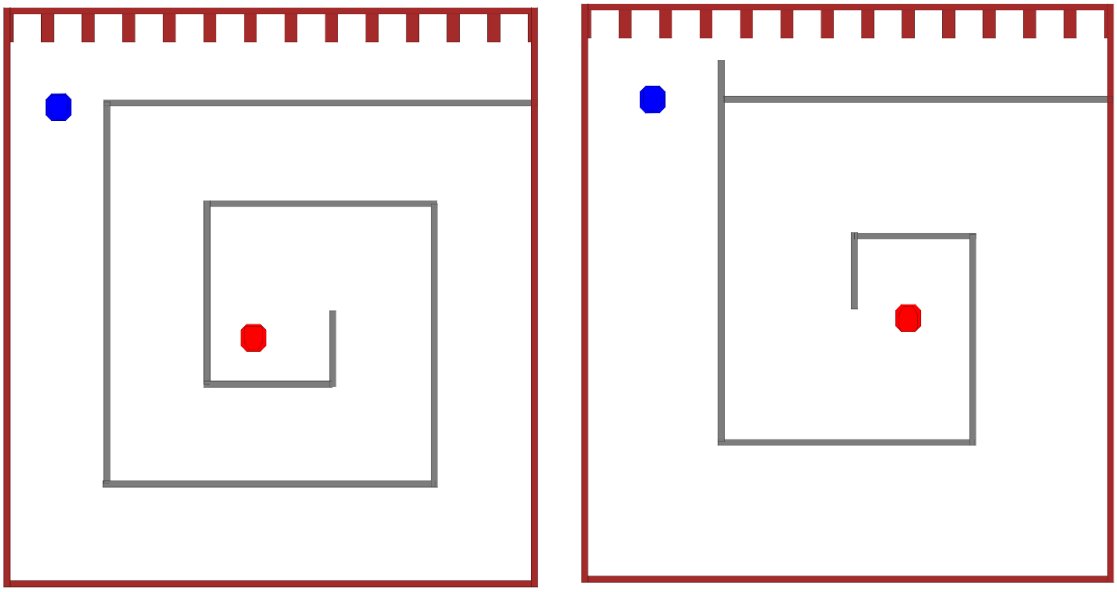


Figure 5.2: Arena 2 (n.p. = 60 cm) on the left and Arena 3 (n.p. = 85.5 cm) on the right. The maps are used for the simulations of the SyRoTek arena for the S1R robot.

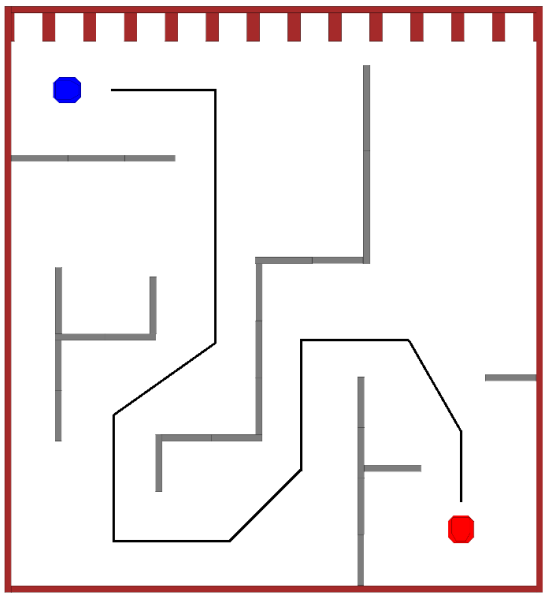


Figure 5.3: Arena 4 (n.p. = 60 cm), used for the simulation of the SyRoTek arena for the S1R robot.

5.2.2 Results

Results are given in the form of a table displaying the most important values for every algorithm. The values are: t_{exp}, t_{min}, t_{max}, $speed_{exp}$ and the success rate. The t_{exp} is the average time in which the robot reached the goal, t_{min} is the minimal time and t_{max} is the maximal time. The $speed_{exp}$ is the average speed of the robot and the success rate is a percentage value of successful runs.

For the simulations the tables present the results for the best configuration of each algorithm on the particular map and the results for the best configuration of the DWA and VFH algorithms overall. The best results are chosen at first by the success rate and then by the average time.

Graphic representation of the data is subjected under each table. Five-number summary is used to get the five most important sample percentiles in the graph. The sample minimum, the lower quartile, the median, the upper quartile and the sample maximum. The results are then discussed in the end of the chapter.

5.2.3 Simulation results for S1R robot

The robots model was restricted to maximum speed 0.3 ms/s, maximum turn-rate 1 rad/s, maximum acceleration 0.05 m/s and maximum turn-rate acceleration 0.05 rad/s. It's size is (length x width): 0.174 x 0.162. The configuration for SND algorithm was already given (tbl. 5.4) and the overall best parameters in the configuration file for the DWA and VFH algorithm were determined as follows in tbl. 5.3, these are the parameters marked as the d*wa best* and *vfh best* in the tables and graphs.

Driver	parameter	value	Driver	parameter	value
DWA	robot_radius	0.1	VFH	cell_size	0.01
	ValAngle	5		window_diameter	20
	ValVelocity	20		safety_dist_0ms	0.09
	ValObstacle	0		weight_desired_dir	5
				free_space_cutoff_0ms	3 000 000

Table 5.3: The overall best parameters from simulations for the DWA and VFH drivers.

Driver	parameter	value
SND	robot_radius	0.06
	min_gap_width	0.14
	obstacle_avoid_dist	0.06

Table 5.4: The best parameters from the simulations for the SND driver.

Arena 0

Arena 0 (Fig. 5.1) simulates the passage through series of narrow passage ways with various direction changes.

Algorithm	t_{exp}	t_{min}	t_{max}	$speed_{exp}$	success rate
dwa	80.842	77.364	91.326	0.1263	100%
snd	80.621	78.812	81.654	0.1247	100%
vfh	153.701	145.762	164.423	0.0775	32%
dwa best	82.693	77.629	108.207	0.1247	100%
vfh best	168.896	155.660	182.886	0.0709	22%

Table 5.5: Results for Arena 0

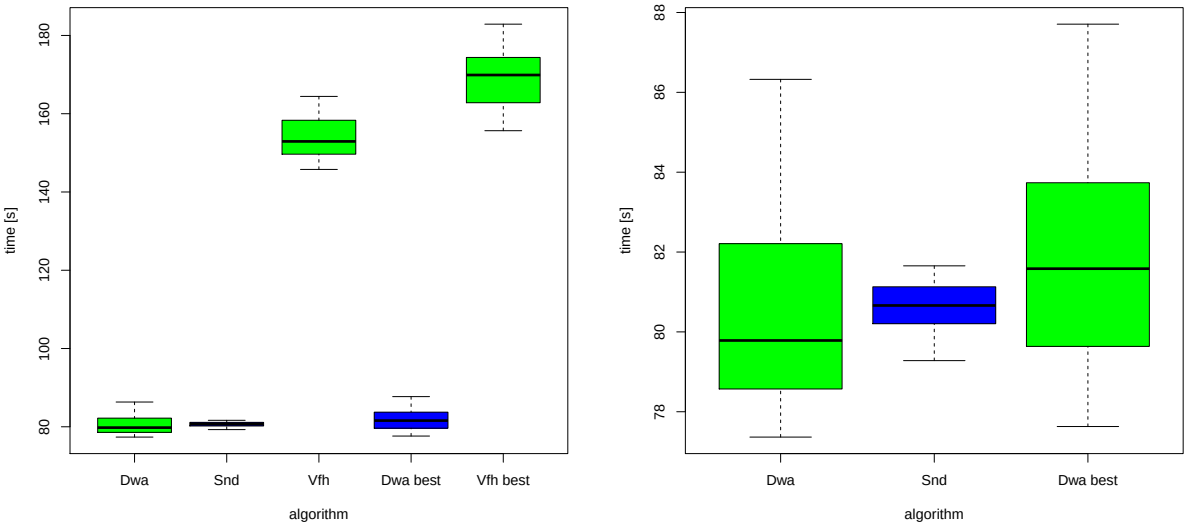


Figure 5.4: Graphic comparison of times achieved by each algorithm. Comparison of all algorithms can be seen on the left picture. To offer more detailed view, the picture on the right shows only the comparison for SND and DWA algorithms.

Arena 1

Arena 1 (Fig. 5.1) simulates a passage through a very narrow corridor. The width of the corridor is less than twice the diameter of the robot. This map has considerably lower success rate than other maps. Thanks to that it was possible to push the navigation algorithms to their limits and see the difference in their reliability.

In order to achieve a higher success rate and thus larger quantity of times for the computation of more reliable data, additional experiments were made on this map, using a different starting point, described in Fig. 5.1. Computed data are thus split into two sets of results.

Algorithm	t_{exp}	t_{min}	t_{max}	$speed_{exp}$	success rate
dwa	268.092	239.299	305.973	0.1153	18%
snd	211.084	208.311	213.881	0.1429	66%
vfh	305.431	289.636	322.115	0.1042	20%
dwa best	253.711	233.581	304.169	0.1228	10%
vfh best	303.762	286.719	324.883	0.1057	14%

Table 5.6: Results for Arena 1 while using the first starting point.

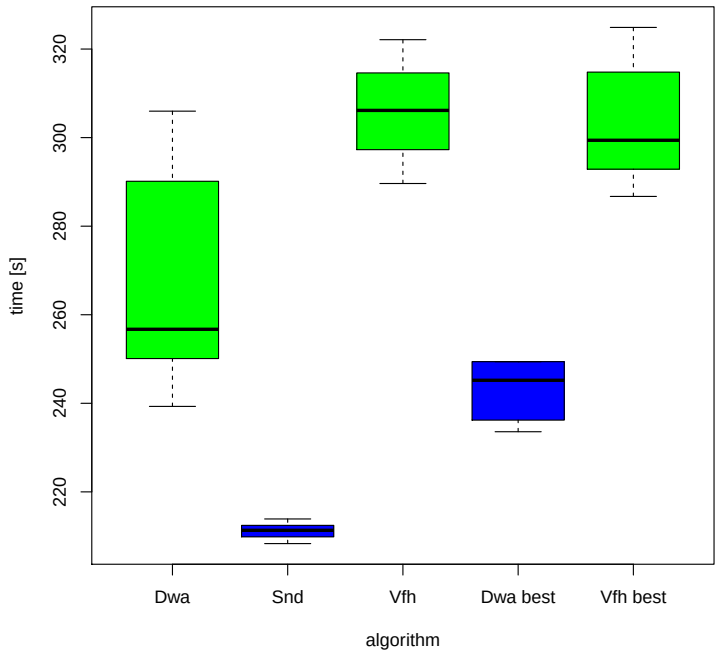


Figure 5.5: Graphic comparison of times achieved by each algorithm.

Algorithm	t_{exp}	t_{min}	t_{max}	$speed_{exp}$	success rate
dwa	146.402	143.614	150.819	0.1160	56%
snd	131.035	127.802	133.538	0.1269	92%
vfh	191.199	177.542	212.584	0.0945	60%
dwa best	146.402	143.614	150.819	0.116	56%
vfh best	189.953	175.381	209.555	0.0946	58%

Table 5.7: Results for Arena 1 while using the second starting point.

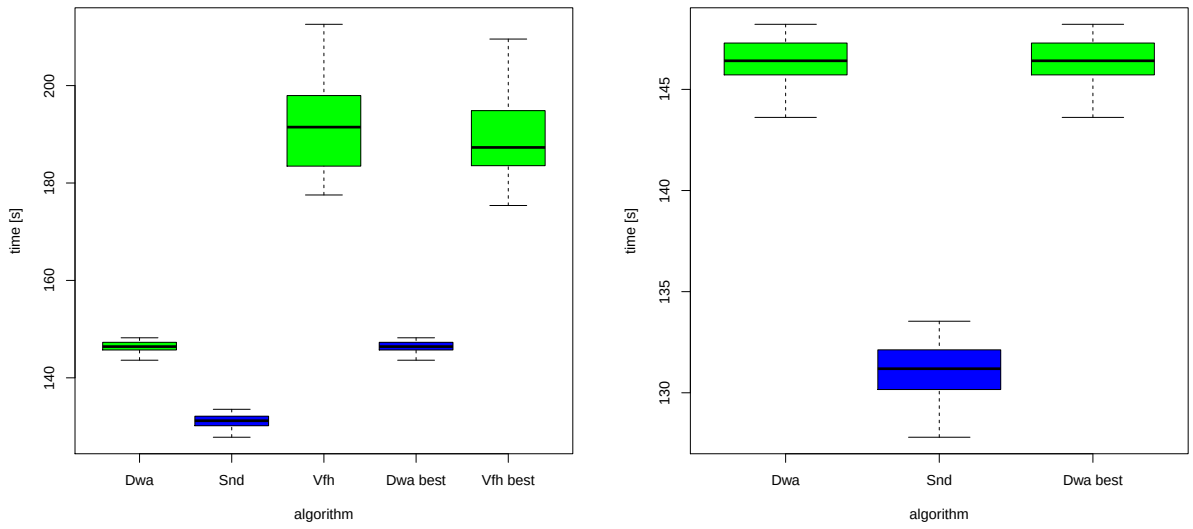


Figure 5.6: Graphic comparison of times achieved by each algorithm. Comparison of all algorithms can be seen on the left picture. On the right picture is only the comparison for SND and DWA algorithms.

Arena 2

Arena 2 (Fig. 5.2) simulates the passage through a wider corridor. The width of the corridor is more than 3 times greater than the diameter of the robot.

Algorithm	t_{exp}	t_{min}	t_{max}	$speed_{exp}$	success rate
dwa	84.189	80.400	88.307	0.1715	100%
snd	85.318	84.542	85.982	0.1656	100%
vfh	115.864	108.242	121.508	0.1425	100%
dwa best	86.129	83.206	89.810	0.1691	100%
vfh best	115.864	108.242	121.508	0.1425	100%

Table 5.8: Results for Arena 2

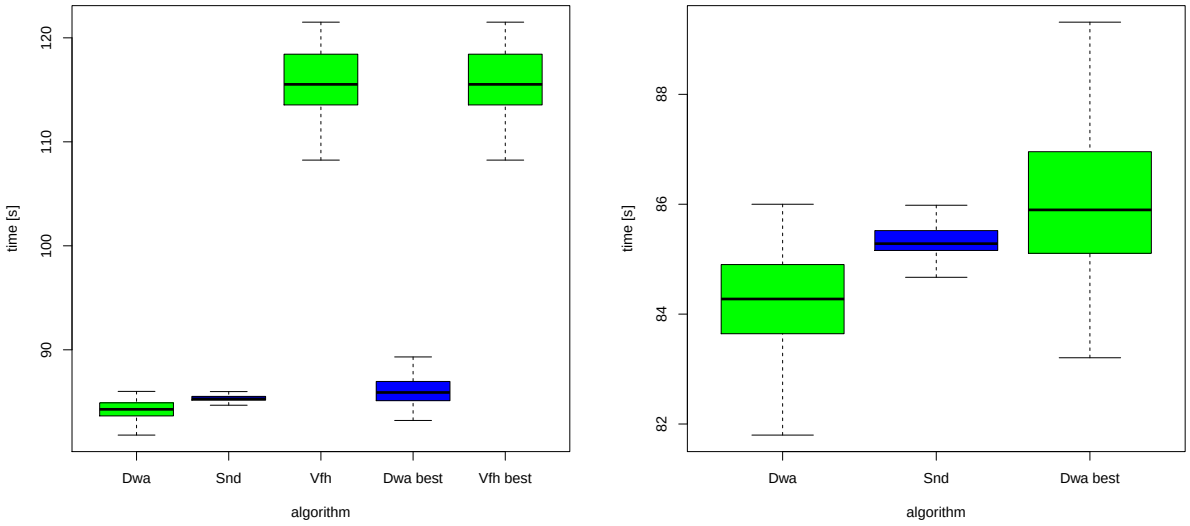


Figure 5.7: Graphic comparison of times achieved by each algorithm. Comparison of all algorithms can be seen on the left picture. On the right picture is only the comparison for SND and DWA algorithms.

Arena 3

Arena 3 (Fig. 5.2) simulates the passage through a very wide corridor. The width of the corridor is more than 5 times greater than the diameter of the robot.

Algorithm	t_{exp}	t_{min}	t_{max}	$speed_{exp}$	success rate
dwa	56.485	54.863	59.866	0.1761	100%
snd	57.641	56.442	58.428	0.1688	100%
vfh	56.976	50.263	67.827	0.2201	100%
dwa best	56.949	55.276	60.168	0.1756	100%
vfh best	56.976	50.263	67.827	0.2202	100%

Table 5.9: Results for Arena 3

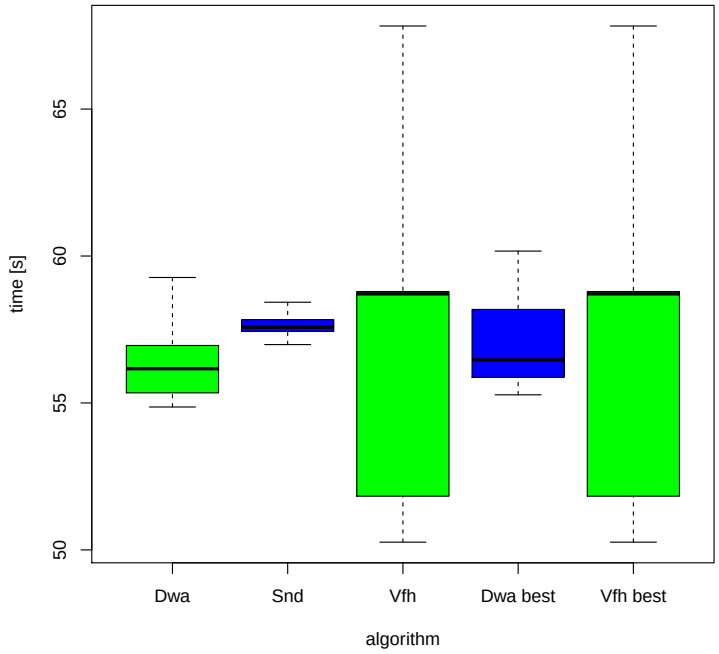


Figure 5.8: Graphic comparison of times achieved by each algorithm.

Arena 4

Arena 4 (Fig. 5.3) simulates the way through a wide passage with several direction changes and open spaces alongside the desired trajectory.

Algorithm	t_{exp}	t_{min}	t_{max}	$speed_{exp}$	success rate
dwa	48.548	46.269	51.362	0.1628	100%
snd	65.310	64.239	66.157	0.1106	100%
vfh	77.051	65.769	120.463	0.1352	80%
dwa best	51.523	47.455	78.900	0.1572	100%
vfh best	73.684	66.366	112.436	0.1382	66%

Table 5.10: Results for Arena 4

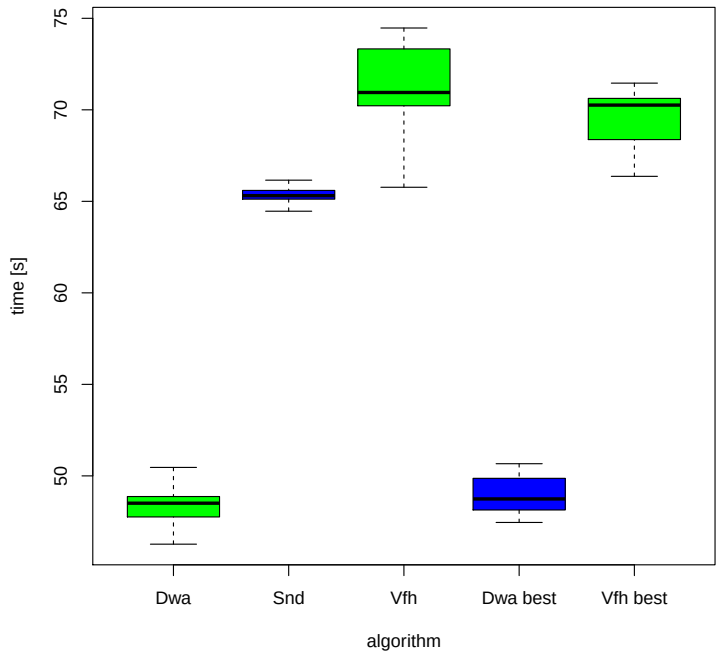


Figure 5.9: Graphic comparison of times achieved by each algorithm.

The DWA algorithm showed exceptional results in this arena. The average time of the algorithm is much better then the time of the other two. The possible explanation could be the fact that the DWA driver didn't slow down before the targets in these simulations.

A function to slow down before the target was implemented after the simulations. Before the implementation of the "slow down" function, it was as if the function was turned off. The

DWA driver now allows the user to turn the option on or off at his will, while the other drivers always slow down as they are closing to the target.

5.2.4 Discussion

The SND algorithm proved to be the most reliable amongst the three algorithms tested. The results form the Arena 1 show a great difference between the success rate of the SND algorithm and the success rate of other two algorithms. This algorithm also proved to have similar time results as the other algorithms on most of the maps.

The DWA algorithm proved to be less reliable, when used in long narrow passages in the Arena 1, aside from this map, its performance was mostly equal or superior to the performance of the SND algorithm.

The VFH algorithm proved to be less reliable than the other two and it's time results were worse than the results of the other two algorithms. It also showed some specific behaviour, which is closely described later in this chapter.

5.3 Real robot - Parameter settings

While several simulations could be performed at the same time, the experiments on the real robot could only be made "one at the time", that made the experiments quite time consuming. Therefore the initial experiments to determine the best configurations were made on a smaller scale than during the simulations. After that 20 test runs were made for three final configurations and for the best configuration from the simulations.

Two sets of the initial experiments were made, a small set of experiments made on the first map and a larger set of experiments on the second map.

Driver	Number of combinations	Parameter	min.	max.	step
DWA	36	robot_radius	0.08	0.12	0.02
		ValAngle	10	20	10
		ValVelocity	10	20	10
		ValObstacle	0	20	10
VFH	32	cell_size	0.01	0.02	0.01
		window_diameter	10	20	10
		safety_dist_0ms	0.05	0.09	0.04
		weight_desired_dir	5	9	4
		free_space_cutoff_0ms	2000000	3000000	1000000
SND	27	robot_radius	0.04	0.08	0.02
		min_gap_width	0.12	0.16	0.02
		obstacle_avoid_dist	0.04	0.08	0.02

Table 5.11: Tested parameters for Map 1.

The initial experiments for the first map for the real robot were made on a small scale. Two test runs were made for each configuration from the table 5.11 for every algorithm and the best three configurations were chosen accordingly to the success rate and the average speed. After that, 20 test runs were made for each configuration and for the best configuration gained from the simulation results.

Driver	Parameter	A	B	C	sim.
DWA	robot_radius	0.08	0.08	0.10	0.10
	ValAngle	20	20	10	5
	ValVelocity	20	10	10	20
	ValObstacle	20	0	0	0
VFH	cell_size	0.01	0.02	0.01	0.01
	window_diameter	20	20	10	20
	safety_dist_0ms	0.05	0.09	0.09	0.09
	weight_desired_dir	9	0	9	5
	free_space_cutoff_0ms	2000000	2000000	3000000	3000000
SND	robot_radius	0.04	0.04	0.04	0.6
	min_gap_width	0.14	0.12	0.14	0.14
	obstacle_avoid_dist	0.04	0.04	0.06	0.06

Table 5.12: Chosen parameters for Map 1.

The initial experiments for the second map were made on a much larger scale than for the first one. Two test runs were made for each configuration from the table 5.13 for every algorithm and the best 10 configurations were chosen for the second round of experiments by the success rate and the average speed. In the second round 10 runs were made for each configuration and the three best configurations with highest success rate and average speed were chosen. In total 20 test runs were made for these 3 configurations and for the best configuration gained from the simulation results.

Driver	Number of combinations	Parameter	min.	max.	step
DWA	48	robot_radius	0.07	0.10	0.01
		ValAngle	10	20	10
		ValVelocity	10	20	10
		ValObstacle	0	20	10
SND	27	robot_radius	0.04	0.08	0.02
		min_gap_width	0.12	0.16	0.02
		obstacle_avoid_dist	0.04	0.08	0.02

Table 5.13: Tested parameters for Map 2.

Configuration	Driver	t_{exp}	success rate	Driver	t_{exp}	success rate
A	DWA	144.776	90%	SND	118.684	100%
B		156.885	90%		126.087	100%
C		165.070	70%		133.976	100%
D		163.327	70%		120.280	100%
E		167.728	60%		130.371	100%
F		173.089	90%		136.411	100%
G		147.012	90%		120.075	100%
H		141.254	70%		137.294	100%
I		148.762	50%		135.443	100%
J		152.518	50%		171.664	90%

Table 5.14: Map 2. Results for the best 10 configurations after 10 test runs.

Driver	Parameter	A	G	B	sim.
DWA	robot_radius	0.08	0.07	0.08	0.10
	ValAngle	20	10	20	5
	ValVelocity	10	10	10	20
	ValObstacle	0	0	10	0

Table 5.15: Chosen parameters for Map 2, DWA driver.

Driver	Parameter	A	G	D	sim.
SND	robot_radius	0.04	0.04	0.04	0.06
	min_gap_width	0.12	0.16	0.14	0.14
	obstacle_avoid_dist	0.04	0.04	0.04	0.06

Table 5.16: Chosen parameters for Map 2, SND driver.

5.4 Experiments with the real robot

For the experiments on a real robot, SyRoTek system was used. SyRoTek system is limited to a smaller area but also offers the possibility to freely rearrange the set up of its arena.

5.4.1 Design

The maps for the real robot were designed with the same concept as the maps in the simulations.

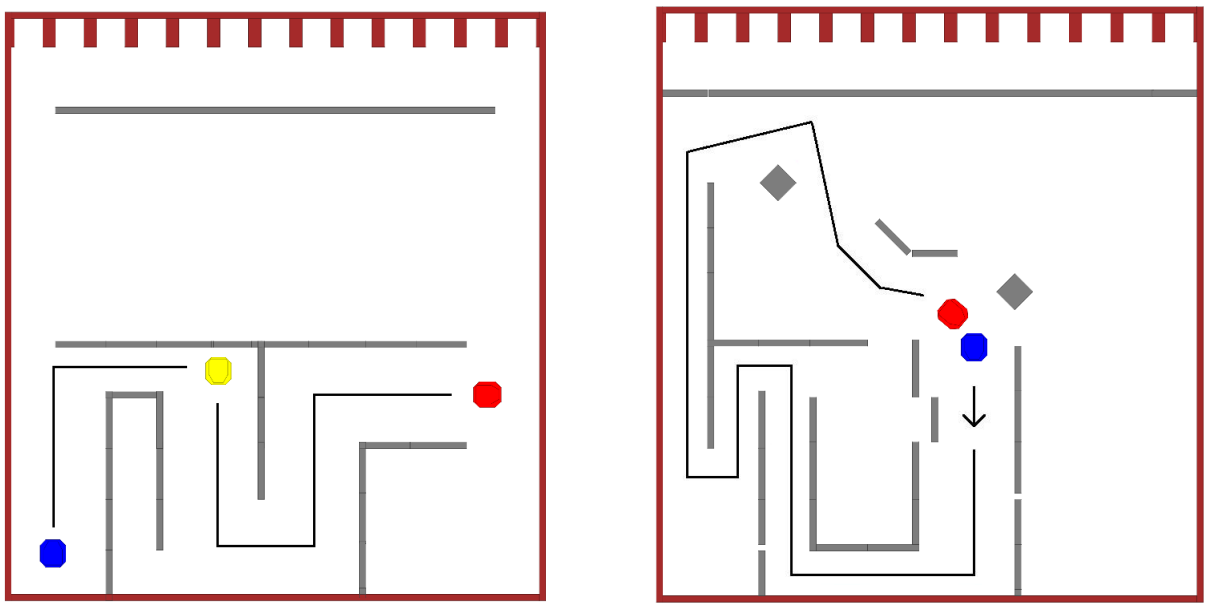


Figure 5.10: Map 1 (n.p. = 28 cm. 60 cm while using the second starting point) on the left and Map 2 (n.p. = 28 cm) on the right. The maps are used for the experiments with a real S1R robot in the SyRoTek system.

5.4.2 Results

For the experiments on the real robot new trial experiments were made, to determine the best possible configurations for each algorithm. The three most suitable configurations of each algorithm were chosen for further experiments together with the best configuration gained from the simulation data. The tables present the results for the best configurations of each algorithm on the particular map and the results for the configuration gained from simulation data.

5.4.3 Results from SyRoTek Arena with a real S1R robot

The robot was restricted to its maximal speed 0.45 ms/s and its maximal turn-rate 1 rad/s.

Map 1

Map 1 (Fig. 5.10) simulates the passage through an environment containing a narrow corridor. Since the VFH algorithm wasn't able to complete the whole map, an additional starting point had to be placed after the narrow part of the passage (see the Fig. 5.10). The data are thus split into two sets of results. The first part is for the first starting point and it contains only the results of the DWA and SND algorithms. The second part contains the results for all the algorithms on a trajectory starting from the second starting point.

Configurations **A** for DWA and **B** for SND driver from the table 5.12 were the best configurations for this map, when the first starting point was used. These are the configurations highlighted in the table 5.17 and showed in the graph 5.11.

Algorithm	Configuration	t_{exp}	t_{min}	t_{max}	success rate
dwa	**A**	53.478	52.171	54.896	100 %
	B	53.860	51.830	56.792	100 %
	C	58.337	55.530	60.868	95 %
	sim	57.8326	55.928	60.781	100 %
snd	A	61.441	59.680	67.977	100 %
	B	60.937	56.670	65.888	100 %
	C	63.321	61.117	68.186	100 %
	sim	63.055	57.268	67.715	100 %

Table 5.17: Map 1. Results for the real robot, while using the first starting point.

Configurations **C** for VFH and **C** for SND driver from the table 5.12 were the best configurations for this map, when the second starting point was used. The best configuration for the DWA driver is the configuration selected from the simulation results. These are the configurations highlighted in the table 5.18 and showed in the graph 5.11.

Algorithm	Configuration	t_{exp}	t_{min}	t_{max}	success rate
dwa	A	33.058	30.561	34.543	95 %
	B	33.456	30.418	35.402	100 %
	C	32.340	30.939	34.405	100 %
	sim	32.321	30.972	33.907	100 %
snd	A	37.602	35.856	39.933	95 %
	B	37.690	36.370	39.404	100 %
	C	37.071	35.358	37.884	100 %
	sim	37.249	35.557	39.390	100 %
vfh	A	37.707	34.508	41.228	100 %
	B	37.502	34.106	40.962	100 %
	C	36.469	25.398	40.708	100 %
	sim	46.867	34.561	182.435	90 %

Table 5.18: Map 1. Results for the real robot, while using the second starting point.

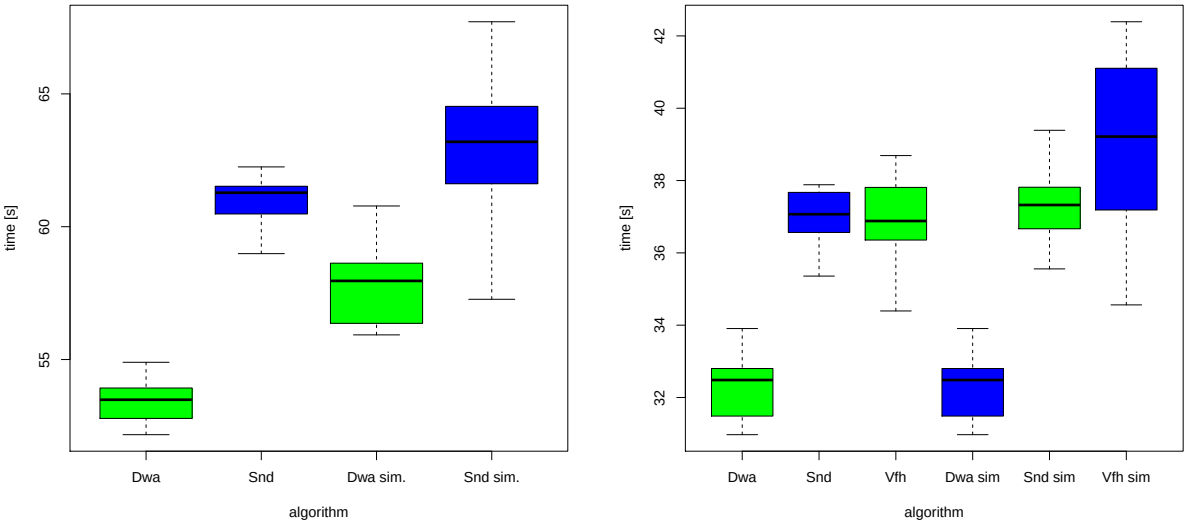


Figure 5.11: Graphic comparison of times achieved by each algorithm. The picture on the left represents the results using the first starting point. The picture on the right represents the results using the second starting point.

Map 2

Map 2 (Fig. 5.10) simulates the passage through an environment containing a lot of narrow corridors. Since the VFH algorithm wasn't able to complete the map, only the DWA and SND algorithms were tested.

Configurations **A** for DWA and **A** for SND driver from the tables 5.15 and 5.16 were the best configurations for this map. These are the configurations highlighted in the table 5.19 and showed in the graph 5.12.

Algorithm	configuration	t_{exp}	t_{min}	t_{max}	success rate
dwa	**A**	143.153	124.298	156.942	95 %
	G	144.866	111.429	197.131	95 %
	B	153.975	131.620	167.875	90 %
	sim	174.318	141.168	206.722	80 %
snd	**A**	119.462	116.555	123.733	100 %
	G	120.788	117.771	129.183	100 %
	D	119.949	117.774	126.956	100 %
	sim	140.640	133.426	147.559	95 %

Table 5.19: Map 2. Results for the real robot.

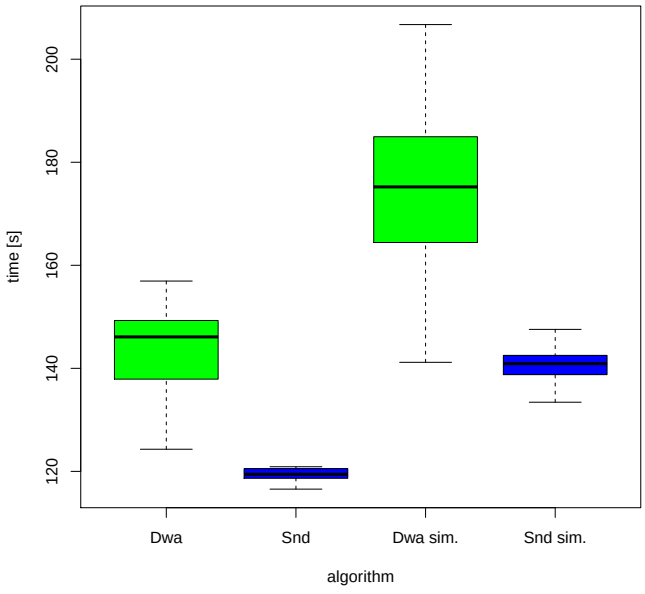


Figure 5.12: Graphic comparison of times achieved by each algorithm

5.4.4 Discussion

The SND driver was more reliable and faster in narrow passages, than the DWA driver. The DWA driver is faster than the other two drivers, while used in environments containing less narrow passages.

When using the real robot, the VFH driver was successfully used only with environments which didn't contain any narrow passages. Therefore such narrow environments were used only for the experiments with the DWA and SND drivers. In wider environments the VFH driver achieved similar results as the other two drivers.

The SyRoTek system doesn't offer information on the current velocity of the robot, so for the experiments this reading was substituted with the desired velocity value from previous iteration. Fortunately this modification didn't seems to have any noticeable impact on the results.

When testing the algorithms on a real robot, few video footages were taken. These videos are available on the CD, presented with the thesis. The footages contain the test runs with the DWA and SND algorithms, using their best configurations for the SyRoTek Arena.

5.5 Discussion of the results

Recommended use for the drivers

The SND driver has proved to be the most reliable both in simulations and experiments. The DWA algorithm had equal results only when used in environments wider than twice the diameter of the robot.

The choice of the algorithm with best achieved times depends on the environment. In narrow environments the SND driver would be the best choice while the DWA algorithm could be recommended for environments with wider passages. The DWA offers an on/off function for slowing down before obstacles. If turned off, the function can increase robots speed on direct trajectories.

Both the SND and DWA algorithms should be easy to use. They have only a few important parameters and since the main parameters are connected with the robot radius, it's easy to determine how to set which parameter, even for an unskilled user.

The VFH driver is inferior to other two drivers in both reliability and speed. Its parameters were originally set for a different robot and the description of the parameters is insufficient for a proper recalibration by someone unfamiliar to the driver.

Comments on the experiments and simulations

- **SND** There were no complications with this driver, it was easy to use and it had no specific functions or characteristics worth mentioning.

- **DWA** A new modification had to be made on the DWA driver for the experiments in the SyRoTek system. Unlike the other drivers the DWA driver requires information on current velocities of the robot. An on/off option was made to substitute the velocity readings by the last velocity command.

- **VFH** While the SND and DWA drivers support a connection with robot's global position, the VFH driver doesn't. Therefore an additional function had to be created in the client file to convert the global coordinates to the coordinates used by the VFH driver. The function is in the client file on the CD provided with the thesis.
  Even after the function was created, the driver had trouble initializing, so every one out of six runs it would load incorrectly and it would have to be restarted. Another modifications would have to be made to correct this error entirely, but that was not necessary for the purpose of this thesis.

  Particular situation occurred with the VFH driver whenever the robot got stuck close to an obstacle and couldn't go as planed, it seemed to start its "escape function". The escape function appears to have a simple system, the robot keeps turning counter-clockwise, looking for an open space in which it could continue it's motion. This function deals with the problem of a robot getting stuck near an obstacle, but has one weakness, the robot always starts turning counter-clockwise regardless of the position of the target. That sometime causes the robot to lose a lot of time, just by turning in the opposite direction than the direction of the target. This function makes it unfavourable for the algorithm to be used in environments containing any narrow turns in the clockwise direction.
  If this function would be fixed to select the turning direction accordingly to the position of the target, it could significantly improve the performance of the VFH driver.

Chapter 6

Conclusion

The goal of this thesis was to compare the results of three local planning algorithms, to develop the Dynamic Window method for the Player system and to find the best possible configurations for these three algorithms for the SyRoTek system.

The Dynamic Window algorithm was successfully implemented as a plugin driver for the Player system and tested both in the Player/Stage environment and on a real robot with the SyRoTek system. The DWA driver proved to be reliable and the robot's movement speed was comparable to other two drivers. It even had superior speed when used in environments with passage ways wider than twice the diameter of the robot.

The next part of the thesis was to find the best parameter configurations for the three tested drivers. These drivers were the DWA driver implemented as a part of this thesis and the SND and VFH drivers from the Player project library. Over 20 000 simulations and 120 hours of experiments with the real robot on the SyRoTek system were made to find the best configurations for the drivers. The results from these experiments will be presented to the Department of Cybernetics at the Czech Technical University in Prague, where the SyRoTek system is placed. Hopefully it will be beneficial for further use of local navigation drivers with this system.
During the process, the DWA and VFH drivers were made operational with the SyRoTek system. The configuration and client files for these drivers were put on the CD, provided with this thesis. The SND driver was already operational with the SyRoTek system.

After the best parameter configurations were found, a set of simulations and experiments was made to gain the results needed for the comparison of the drivers. The comparison proved the SND driver to be the most reliable one. The SND driver also achieved the best time results in narrow environments. The DWA driver proved to be slightly inferior to the SND driver in narrow environments, but in wider environments it proved to have equal results in reliability and equal or superior time results. The VFH driver proved to be insufficient for the use in narrow environments. It doesn't offer any advantages over the other two drivers and it's use was accompanied with several difficulties.

A website has been launched for the DWA driver [13]. The website contains information about the driver, detailed manual and a download section with the project files. As the next course of action, the developed DWA driver will be offered to the Player/Stage project. If approved by the Player project, the DWA driver will become a part of the standard Player distribution and information from this website will be moved to the official website of the Player project.

Bibliography

[1] M. Seder, K. Maček, I. Petrovič, *An integrated approach to real-time mobile robot control in partially known indoor environments*. Industrial Electronics Society, 2005. IECON 2005.

[2] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. *The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems*. pages 317-323, Coimbra, Portugal, July 2003.

[3] M. Kulich , J. Chudoba , K. Kosnar , T. Krajnik , J. Faigl and L. Preucil *SyRoTek—Distance teaching of mobile robotics*, IEEE Trans. Educ., 2013

[4] S. M. LaValle, *PLANNING ALGORITHMS*. Published by Cambridge University Press in 2004.

[5] J. Borenstein and Y. Koren, *The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots*. IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, 1991.

[6] I. Ulrich and J. Borenstein, *VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots*. IEEE International Conference on Robotics and Automation, 1998.

[7] J. Minguez and L. Montano, *Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios*. IEEE Transactions on Robotics and Automation, 2004.

[8] J. W. Durham and F. Bullo, *Smooth Nearness-Diagram Navigation*. Intelligent Robots and Systems, 2008. IROS 2008.

[9] D. Fox, W. Burgard and S. Thrun, *The Dynamic Window Approach to Collision Avoidance*. Robotics and Automation Magazine, IEEE, 1997.

[10] The official website of the Player Project.
http://playerstage.sourceforge.net/

[11] The official website of the SyRoTek system.
https://syrotek.felk.cvut.cz/

[12] B. Petersen and J. Fonseca, *Writing Player/Stage Drivers, a howto for ERSP Player Driver Source Package*. From the project: Player/Stage - Player driver implementation for ERSP, 2006.

[13] The website launched for the implemented DWA driver.
http://imr.felk.cvut.cz/dwa/

List of Figures

List of Tables

Appendix

CD Content

In table 6.1 are listed names of all root directories on CD

Directory name	**Description**
DWA driver	Contains the newest version of the DWA driver. The versions in other folders are the versions used during the activities connected with their folder location.
Results	Database files from the simulations and text files from the experiments on the SyRoTek system.
Simulation files	Maps, world files, client files, configuration files and the dwa driver files used for the simulations.
SyRoTek files	Maps, client files, configuration files and the dwa driver files for the SyRoTek system.
thesis.pdf	The pdf file containing this thesis.
Thesis	Source files for this thesis.
Video	Videos from the experiments in the SyRoTek system.

Table 6.1: CD Content